# Web Application Security
## Boulder Linux Users Group

**Ben Whaley**

**September 13, 2007**

## Agenda Summary

- Security Warm up
- Common input exploits
- Secure coding with input validation
- Self-assessment and testing

… if time permits …

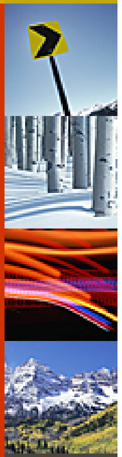
- Hot topics in application security

## What is security?

- **What is security?**

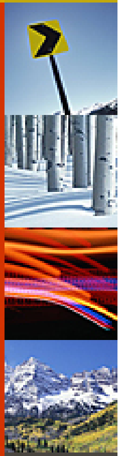
- **3 risks and 3 priorities:**
  - **Disclosure -> Confidentiality**
  - **Corruption -> Integrity**
  - **Unavailability -> Availability**

- **Multi-layered defense**
  - **We *have* to deal with application and host security**



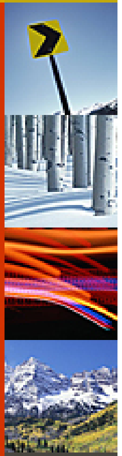
## Open Web Application Security Project

- The OWASP guide is the de-facto authoritative resource for web application security
  - For example, the PCI DSS standard requires that applications are developed according to OWASP
- Too “loose” to be called a standard, but still a wonderful resource
- Lots of resources:
  - OWASP Guide
  - Top 10 Lists
  - WebGoat training application
  - WebScarab
  - ...and more!



## Agenda Summary

- Security Warm up
- Common input exploits
- Secure coding with input validation
- Self-assessment and testing
- Hot topics in application security



## Input Exploits

- **External input to application may contain special characters**
  - **Various characters have special significance to the database, or the web/application server, or perhaps the OS**

- **Untrusted input can come from:**
  - **URL parameters**
  - **Form elements**
  - **Cookies**
  - **Database queries**
  - **Other programs!**

- **AKA: Command injection**

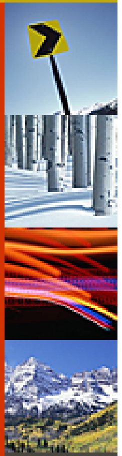
## SQL Injection attacks: The Basics

- **Four main types of attacks**
  - **SQL manipulation**
  - **Code Injection**
  - **Function call injection**
  - **Buffer overflows**

- **Most databases engines are susceptible to the first two categories (MS SQL, MySQL , PostgreSQL, Oracle, DB2…)**

- **The last two are more Oracle specific and not as widely published**

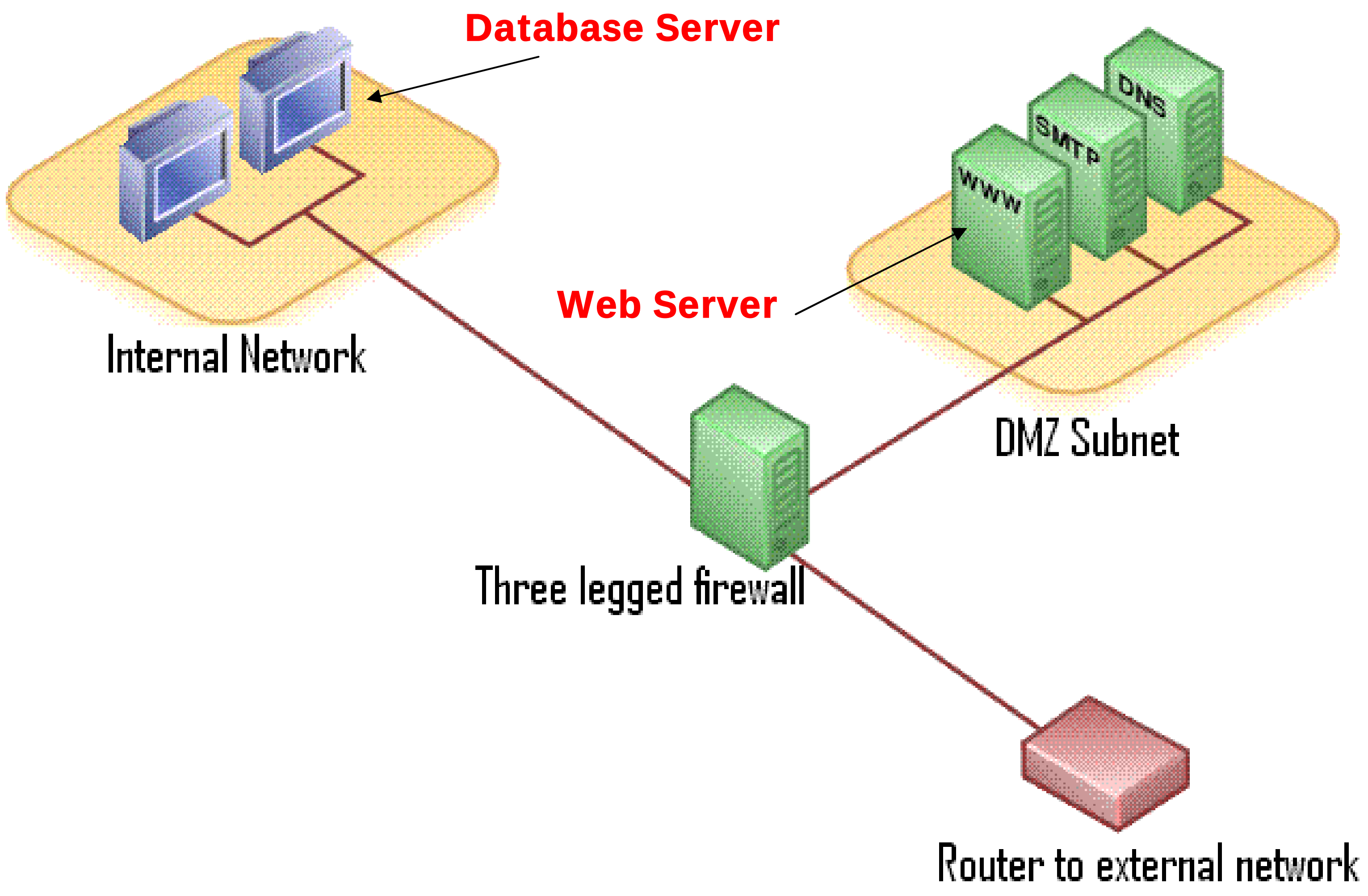
## SQL Manipulation and code injection



- **SQL Manipulation**
  - **By far the most common attack**
  - **Modify variables passed to the WHERE clause of a query to always return TRUE**
  - **Usually accomplished by passing unexpected characters that SQL interprets literally**

- **Code injection**
  - **Involves executing multiple SQL statements at once**
  - **MySQL natively supports this. Other databases (Oracle) do not.**

## Simplified Web Application Architecture



-- Compliments of Wikipedia

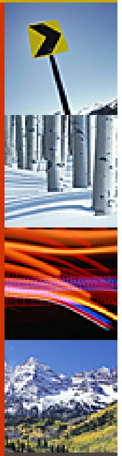
## Simplified Authentication Mechanism

- PHP accepts credentials from the user via POST parameters
- Opens connection to MySQL
- A SELECT statement attempts to match the input against the database
- If a match is found, the user is authenticated
- If not, the log in fails

## Hands on

- What might this look like in PHP?
- Looks great, except...

## SQL Manipulation Example



- **An attacker can pass SQL commands as input variables**
- **For example:**
  - If the attacker set `Username` to: `admin' OR '1'='1`
  - And password to: `anything`
- The SQL statement becomes:

```
SELECT * from user WHERE user = 'admin' OR '1'='1’ AND
   pass = 'anything'
```

- **Admin is logged in without providing a password!**

## Code Injection Example

- **From the SQL Manipulation example:**

  ```
  SELECT * from auth WHERE user = '$username' OR pass
    = '$password'
  ```

- **Set** `username=anything` **and** `password=blah' OR '1'='1'; use mysql; UPDATE user SET PASSWORD=password('blah') where user='root'; FLUSH PRIVILEGES; use blug; SELECT * from auth where user='`

- **The full query becomes:**

```
select * from user where user=‘anything’ and
   pass='blah' OR '1'='1'; use mysql; UPDATE user SET
   PASSWORD=password('blah') where user='root'; FLUSH
   PRIVILEGES; use blug; SELECT * from auth where
   user='
```

- **We're off the hook - PHP's mysql_query() function does not support this syntax**

## Input Exploits: Cross-site scripting

- **Two general types of XSS:**
  - **Reflected – Attack occurs when code is returned from the server (search results, error messages, etc)**
  - **Persistent – data stored permanently, may affect many users**

## Cross-site Scripting Example: Reflected

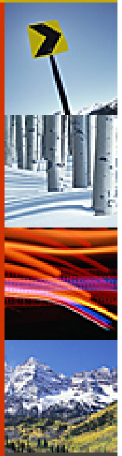
- **A popular web site requiring user registration displays a greeting with data from the URL query string to the user**
  - **i.e., visiting** `http://www.example.net/index.php?user=ben` **results in "Welcome, ben" on the front page**

- **Attacker sends email to a user of example.com, embedding javascript in the URL:**

  -
    ```
    http://www.example.com/index.php?user=
    <script>document.location='http://www.example.com
    /cookie.cgi?' +alert('hahaha!')</script>
    ```

- **Prays on the user's legitimate trust for you SSL-protected site**

## Cross-site Scripting Example: Persistent

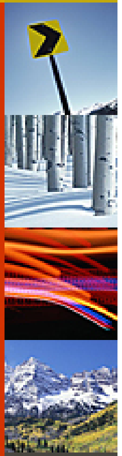


- **Consider a bulletin board application**
  - **Users post “threads” for others to view**
  - **The application stores authentication session information in the cookie (a common practice)**
  - **A malicious user includes the following text in his post:**

```
<script>document.location='http://www.example.c
  om/cookie.cgi?' + alert('hahaha!')</script>
```

## Agenda Summary

- Security Warm up
- Common input exploits
- Secure coding with input validation
- Self-assessment and testing
- Hot topics in application security



## Secure Coding with Input Validation

- **Defining input: All forms of input data to a program, obtained from a user, another program, a database, or any other external entity.**

- **Protecting against input attacks**
  - **Validate all input**
  - **Confirm data integrity**
  - **Verify data “realism” (i.e. business rule correct)**

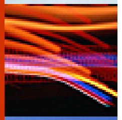
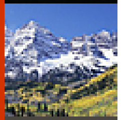
## Types of validation: Positive Validation

- **Positive validation: Check for known good values.**
- **Characteristics:**
  - Reject all values that don't meet tight constraints
  - Strongly typed
  - Length checked
  - Range check (if applicable)
  - Unsigned (if applicable)

- **Pseudo-example: Accepting a social security number**

```
unsigned int SSN = 0
If SSN != [0-9]{3}-[0-9]{2}-[0-9]{4}
   Then error “Sorry, this is not an SSN.”
Else
   INSERT INTO cSSN values SSN;
```

## Types of validation: Negative Validation

- **Negative validation: Check for known bad values.**
- **Characteristics:**
  - Define and reject invalid data
  - Requires never-ending maintenance of “bad” values
- **Example:**

```
unsigned int SSN = 0
Bad_values = “<‘!?>”
If SSN contains Bad_values
  Then error “Sorry, this is not an SSN.”
Else
  INSERT INTO cSSN values SSN;
```

## Types of validation: Sanitization

- **Sanitizing data: Escape and translate data to safely capture and process the input.**
- **Characteristics:**
  - **Allow all data**
  - **Use character encodings or escapes to “sanitize” potentially harmful characters**
  - **Requires care and feeding**
- **Example:**

```
unsigned int SSN = 0
Bad_values = “<‘!?>”
If SSN contains Bad_values
  Then SSN = sanitize(SSN)
INSERT INTO cSSN values SSN;
```

- In PHP, use addslashes()

## Securing our PHP application

- **Positive Validation**

```
# Allowing only alphanumerics and the underscore
# NOTE: No strong password support!
$permit = '/^\w+$/';
if (!preg_match( $permit, $username)
    || !preg_match( $permit, $password)) {
    echo "Error: Only letters and numbers permitted.<br>";
    exit;
}
```

- **Sanitization**

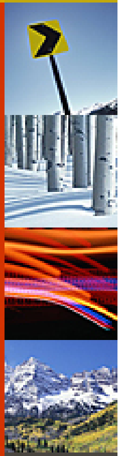
```
$username = addslashes( $_POST['username'] );
$password = addslashes( $_POST['password'] );
```

## Validation: Where to do it

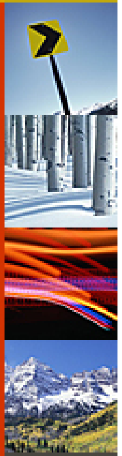
- **From the user's perspective, client-side validation is slickest**
  - **Typically using javascript**
  - **User doesn't have a wait for a page reload/re-render**
- **Unfortunately, attackers can bypass all client-side validation**
  - **So we *must* do it on the server**
  - **Client-side validation is a second priority**
- **Always validate before the value is used**

## Agenda Summary

- Security Warm up
- Common input exploits
- Secure coding with input validation
- Self-assessment and testing
- Hot topics in application security



## Assessment and Testing: Input Validation



- **Parameter manipulation with a local proxy server**
  - Proxy servers intercept request and forward it on behalf of the client
  - Allows control over destination, content, etc.
  - Supported by all major browsers
  - A local proxy allows the developer to view raw requests, manipulate HTTP requests, and more
- **Automated testing**
  - Fuzzing is providing randomized input, or fuzz, to an application
  - Using a preset rules database, thousands of inputs can be tested at a time
  - Warning: Only use in development or test environments!

## Proxy servers

- **What is a proxy server?**
  - **“Site” proxies are commonly used to filter and control web traffic**
  - **All outgoing traffic to port 80 and/or 443 can be forwarded to the site proxy**
  - **Squid, bluecoat, etc do this**
- **What is a local proxy?**
  - **Rather than a site-wide server that intercepts all HTTP traffic, a local proxy is installed on YOUR desktop**
  - **The web browser is pointed at the local proxy port (for example, localhost port 8080)**
  - **The local proxy server then receives all HTTP requests and responses before they are sent to the server and browser.**

## Popular local proxies

- **Paros Proxy**
  - **Simple to turn on/off request and response “trapping”**
  - **Manipulating data is a piece of cake**
  - **Has a spider to map the web site hierarchy for you (with cookie support)**
  - **Filter support**
  - **Free!**
- **WebScarab**
  - **Portable (Written in Java)**
  - **SSL support**
  - **Beanshell – arbitrarily complex Java request manipulation**
  - **Built-in parameter fuzzer**

## Input Fuzzing

- Relatively recent tool for testing application security
- Can test any type of input!
  - Network protocols
  - URL parameters
  - HTML form inputs
  - …
- Lots of frameworks out there! Such as:
  - SPIKE Proxy
  - WebScarab
  - Peach fuzz
- Many are incomplete, complex, or abandoned

## Input fuzzing with WebScarab

- **WebScarab fuzzes parameters, defined as:**
  - **Part of a path. Ex:** `www.example.com/some_path` **(some_path= path parameter)**
  - **URL Query parameter. Ex:** **`http://example.com/index.html?username=admin`** **(username)**
  - **Cookie parameter Ex:** `Cookie: lang=en-us; ADMIN=no; y=1 ; time=10:30GMT ;` **(All of lang, ADMIN, y, and time)**
  - **POST parameters. Any HTML form that POSTs input (content-type must be set to application/x-www-form-urlencoded, which is most forms)**

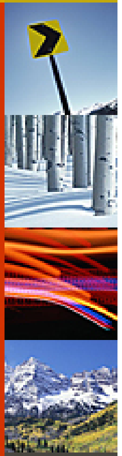
## Hands On

- **Testing with Microsoft Fiddler**
- **Input fuzzing with SPIKE Proxy**

## Agenda Summary

- Security Warm up
- Common input exploits
- Secure coding with input validation
- Self-assessment and testing
- Hot topics in application security

## Accepting Incoming Email



- **Spammers (and other attackers) are actively harvesting email addresses from web pages**
  - **Many automated tools to scan a site and report mailto: links**
  - **So, we pretty much have to stop using them**

- **Replace all mailto: links with form-based mail submission forms**
- **Of course, be sure the form submission application is secure**
  - **Almost always, this means something needs to be hardcoded**
  - **Usually this is the "to" address… sometimes the message**

## Preventing Automatic Form Submission

- **Automated form submission has brought spam to the web!**
- **There are many tools in our arsenal, but they are a wonderful example of trading convenience for security**
  - **The last thing we want to do is make it too hard for people to use our web form!**
  - **However, form spam can bury useful communications anyway**
  - **In some cases legislation regulates what we can use (Section 508 in the US Rehabilitation Act)**

## Preventing Automatic Form Submission

- **Some solutions:**
  - CAPTCHA: Completely Automated Public Turing test to tell Computers and Humans Apart
  - KittenAuth/HumanAuth
  - Sessions
  - JavaScript
  - Style Sheets
  - Key Words
- **Some issues:**
  - User acceptance
  - Section 508 compliance
  - False positives
  - Server load
  - Client compatibility

## Web 2.0 Security

- The same security vulnerabilities and controls apply to AJAX sites
- However, AJAX often requires additional or stronger controls because they are usually complex, bidirectional, and asynchronous
- AJAX applications often have weak authentication, session management, and error handling

## AJAX Injection

- **The heart of AJAX is the XMLHttpRequest**
  - **Allows for asynchronous server communications and browser updates**
  - **(Originally developed by Microsoft!)**
- **The browser can be updated with just simple HTML (DOM), XML, or another structured data format**
- **These XMLHttpRequest calls are just normal HTTP requests**
  - **They require all the same authentication and session management controls normal HTTP requests do**
  - **That's right, must authenticate EVERY request!**

## That's All, Folks

**Thanks!**

**ben at atrust dot com**

**Applied Trust is hiring! See our jobs page:**

**http://www.atrust.com/company/jobs**